

# **Understanding Self-healing in Service Discovery Systems**

**Chris Dabrowski and Kevin Mills**

**WOSS 2002**

**Charleston, South Carolina**

**November 18, 2002**

# Observations on Self-healing in Distributed Systems



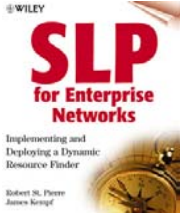



- Recovery strategies are critical for self-healing as failure rate increases.
  - More so than other factors (e.g., architecture, topology, consistency-maintenance mechanisms)
- Recovery strategies can interact in complex and unexpected ways
  - Redundancy (only one is necessary)
  - Complimentaryness (both are necessary)
  - Interference (one strategy prevents another from succeeding)
- When designing self-healing distributed systems based on service discovery protocols, need to consider:
  - The types of failure expected and their likelihood
  - Detailed protocol behaviors (e.g., discovery, update propagation, recovery) and not simply the application-programming interface.

# Dynamic discovery protocols in essence...

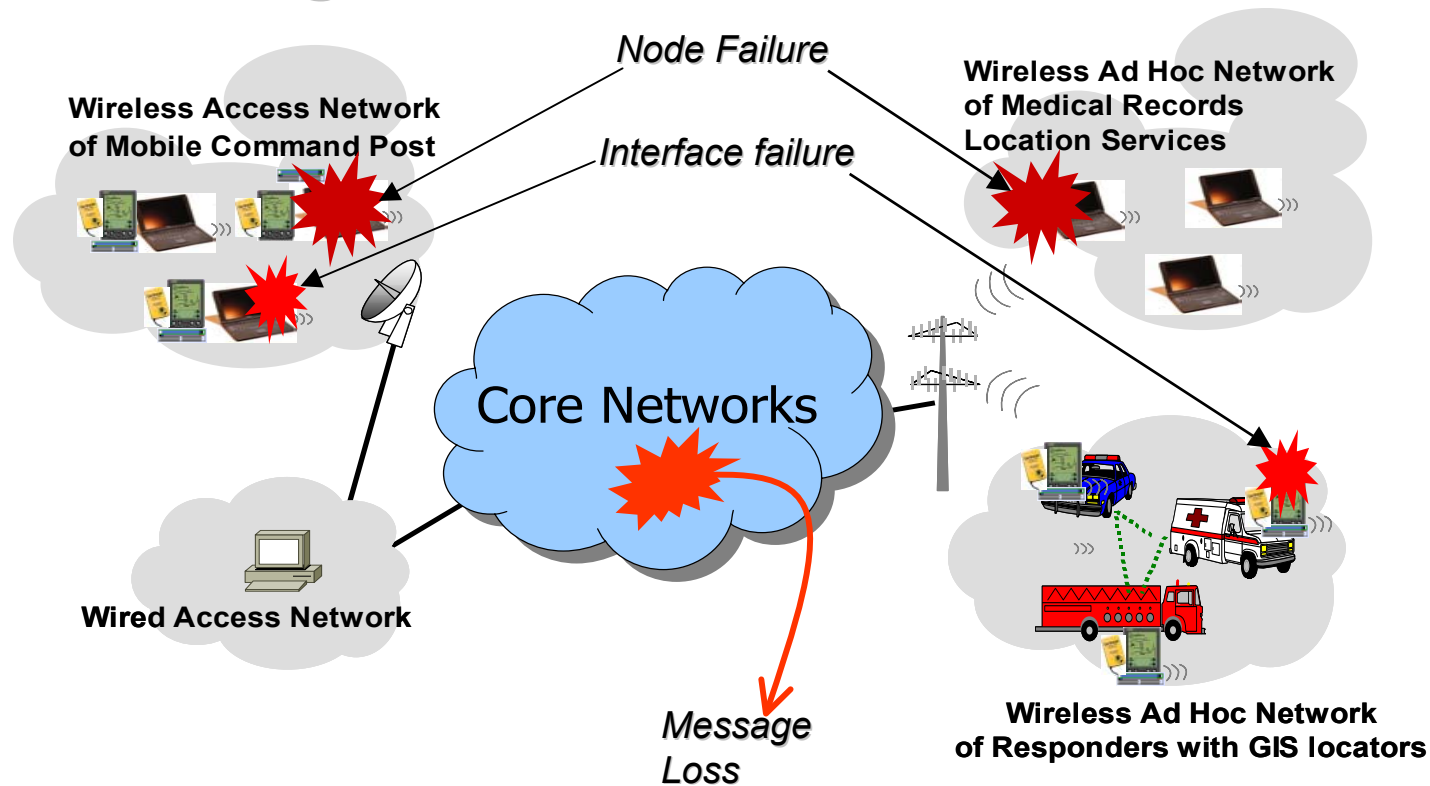
enable ***distributed software components***

- (1) to ***discover*** each other without prior arrangement,
- (2) to ***express*** opportunities for collaboration,
- (3) to ***compose*** themselves into larger collections that cooperate to meet an application need, and
- (4) to ***detect and adapt*** to failures.

Some examples:

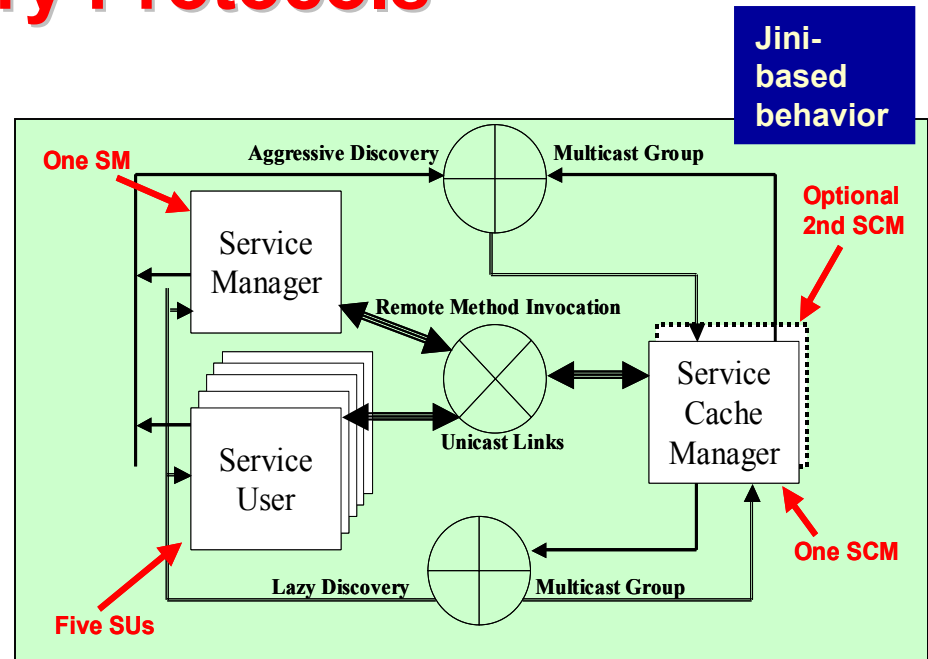
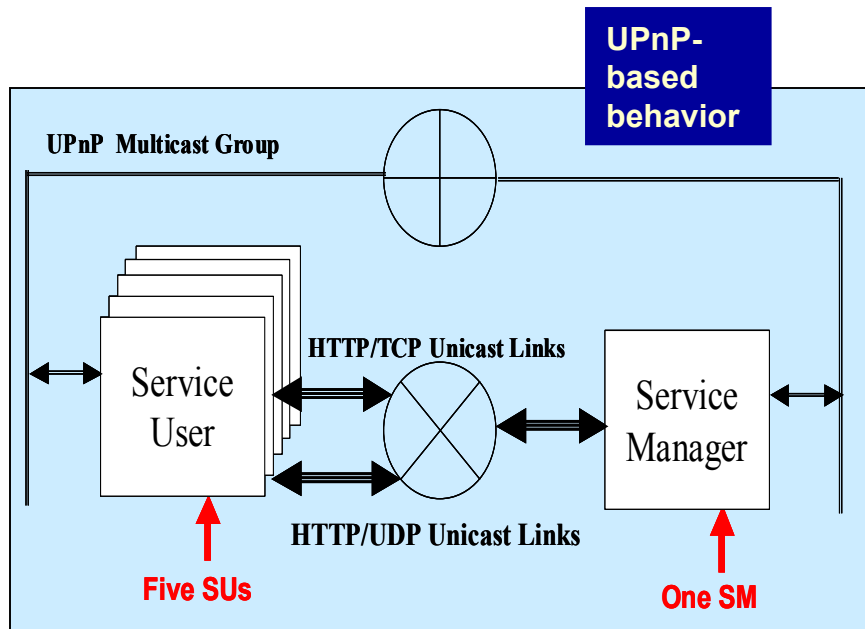
 <b>3-Party Design</b>	 <b>2-Party Design</b>	 <b>Adaptive 2/3-Party Design</b>
 <b>Vertically Integrated 3-Party Design</b>	 <b>Network- Dependent 3-Party Design</b>	 <b>Bluetooth™ Network-Dependent 2-Party Design</b>

# Self-healing in Hostile and Volatile Conditions



- Service discovery systems must ensure consistency of information about services in failure environments
- Contributing factors: recovery strategies, architectures, topologies, and consistency-maintenance mechanisms (polling & notification)
- *This study focuses on role of recovery strategies.*

# Two Generic Architectures Underlie Six Discovery Protocols

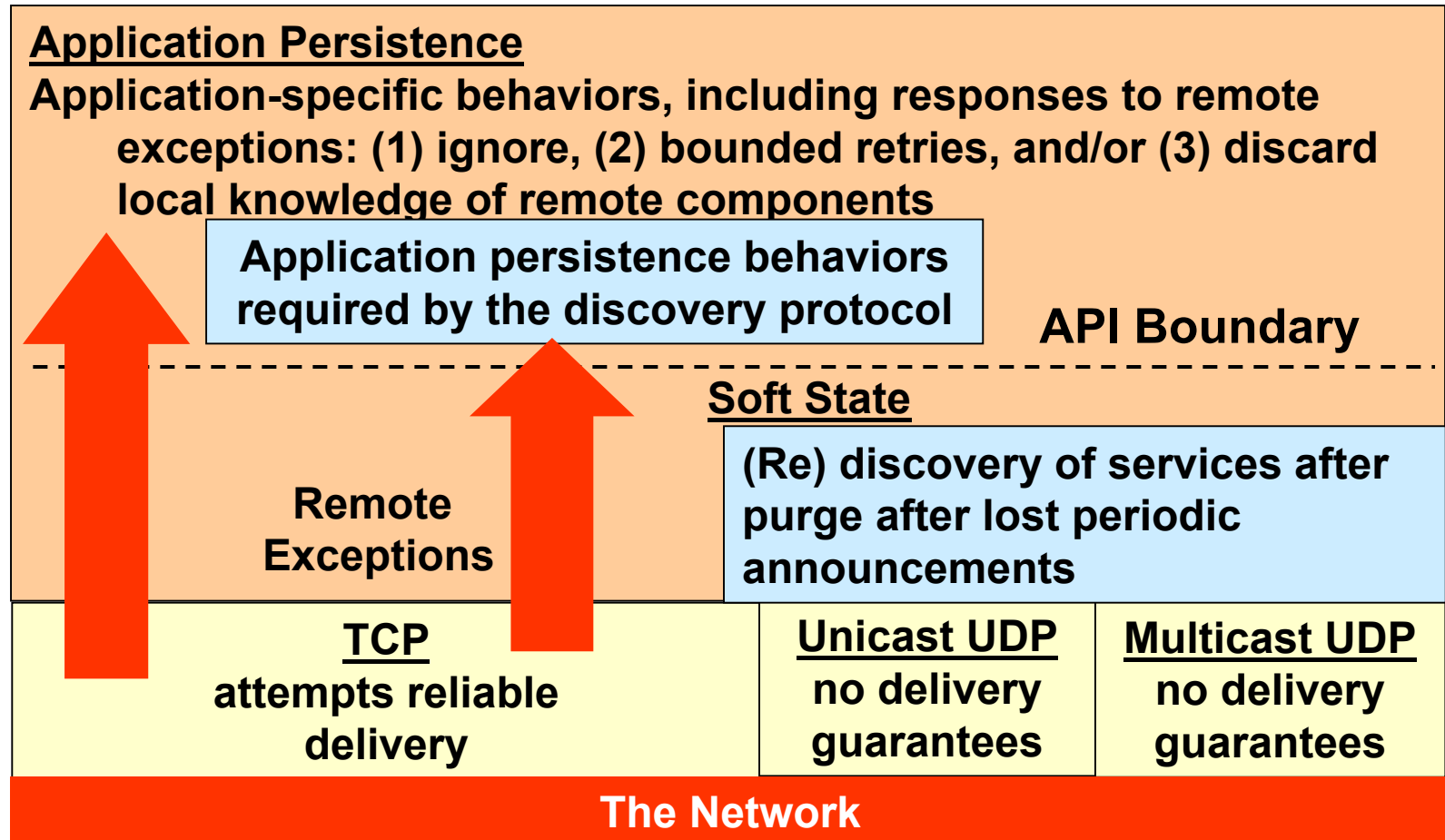


## Update Propagation Method

- **Notification** – Updates forwarded by Managers immediately after they occur.
  - ✓ Service Users request leases with Service Managers to obtain notifications
  - ✓ Notifications rely on TCP for robustness, but TCP may fail and issue a remote exception

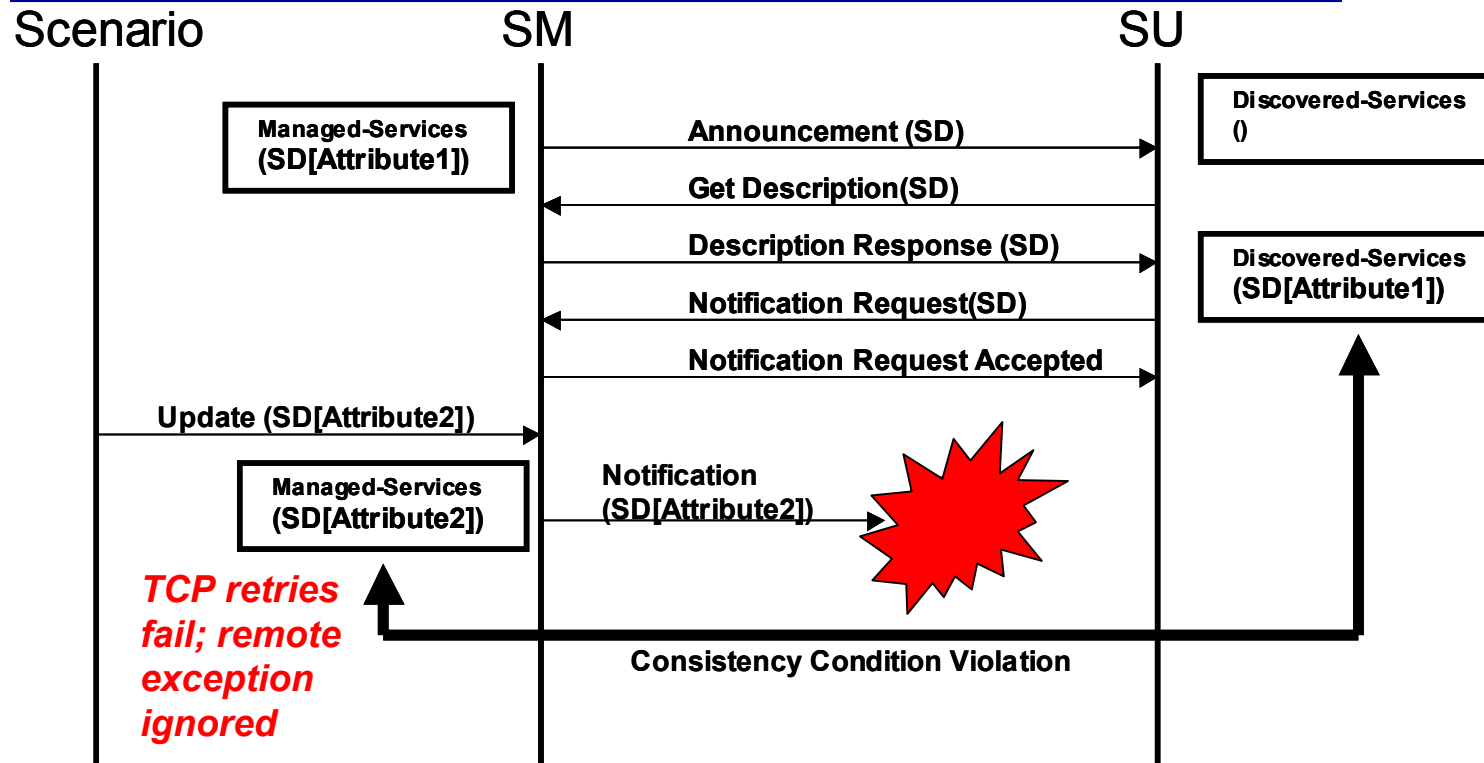
# Understanding Contribution of Failure Detection and Recovery Strategies to Update Effectiveness

## Types of Strategies:



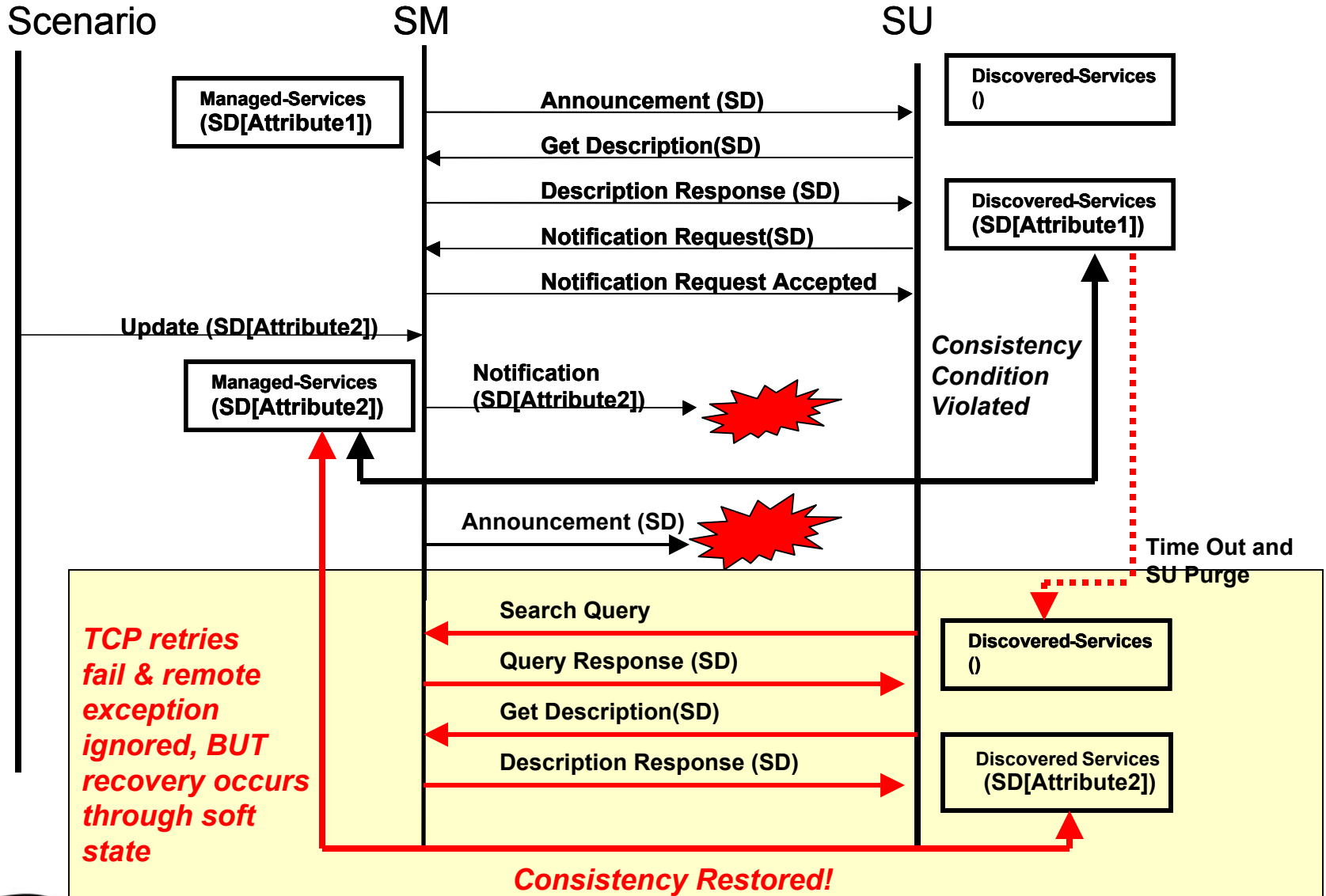
# Consistency Maintenance Using Notification

*For All (SM, SU, SD):  
(SM, SD [Attributes1] ) IsElementOf SU discovered-services  
SD [Attributes2] IsElementOf SM managed-services  
implies Attributes1 = Attributes2*



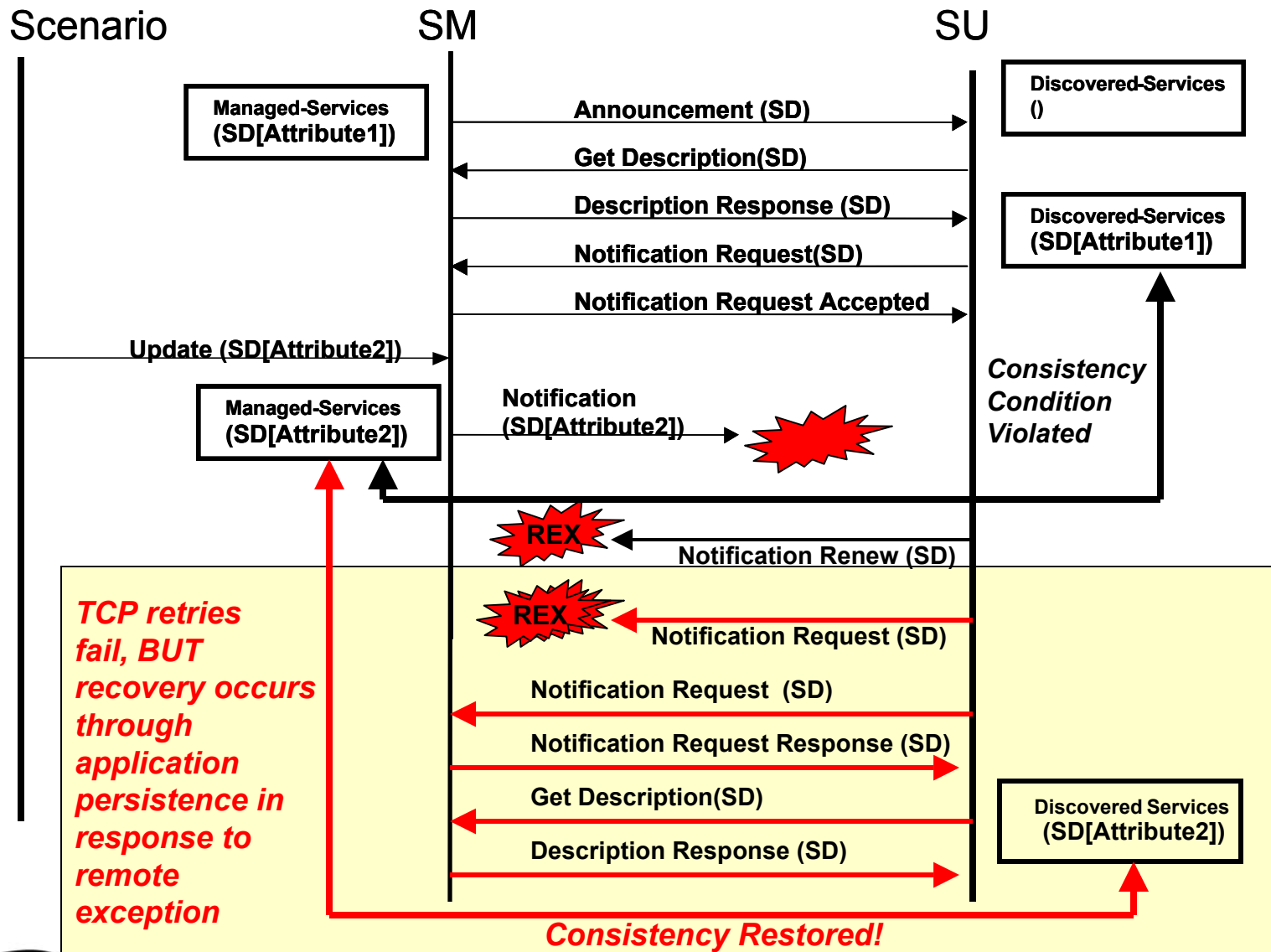
**How well does the system restore consistency after failure?**

# Soft State Recovery of Service After Failed Notification

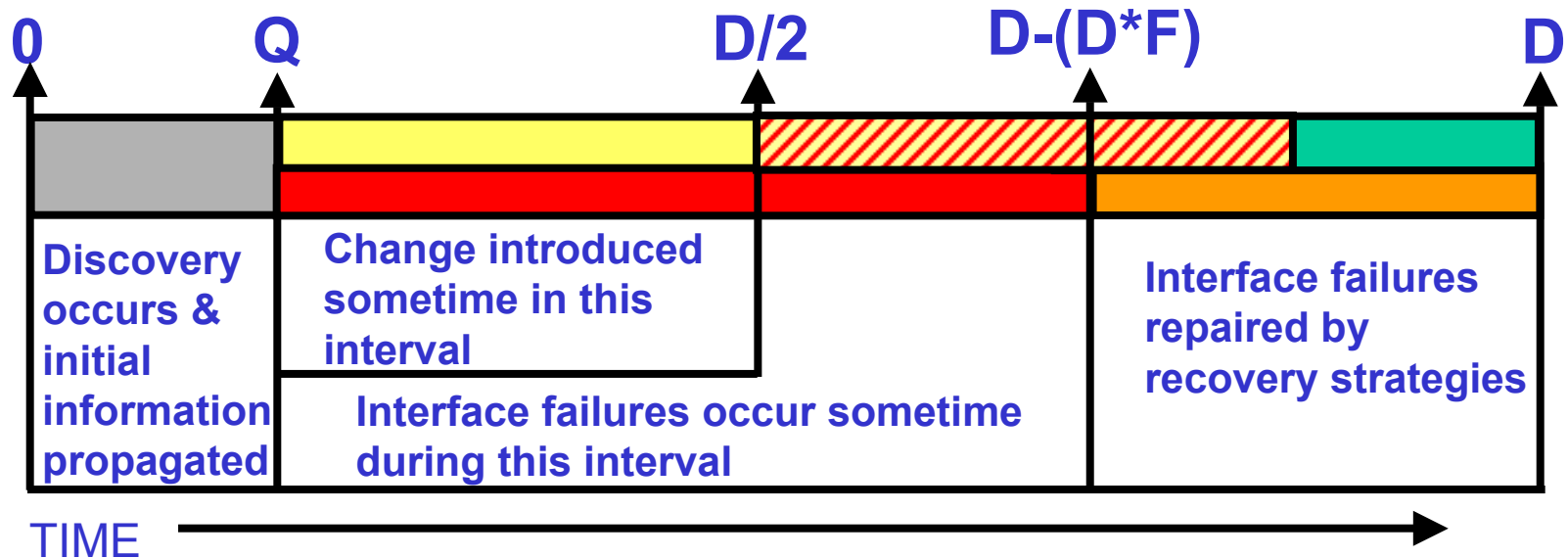




# Application Persistence Recovery of Service After Failed Notification



## Interface-Failure Model for Experiment



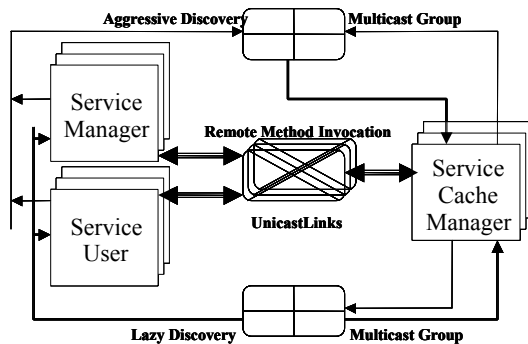
- Random Processes
1. Choose a time to introduce the change [uniform( $Q$ ,  $D/2$ )]
  2. For each node, choose a time to introduce an interface failure [uniform( $Q$ ,  $D-(D*F)$ )]
  3. When each interface failure occurs, choose the scope of the failure, where each of [Rx, Tx, Both] has an equal probability

$Q$  = end of quiescent period (100 s in our experiment)

$D$  = propagation deadline (5400 s in our experiment)

$F$  = Interface Failure Rate (variable from 0% - 75% in 5% increments in our experiment)

# Modeling and Analysis Approach

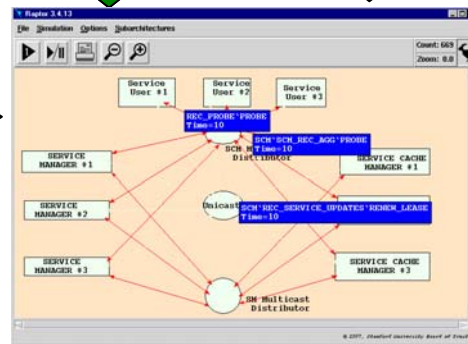


Behavior Model

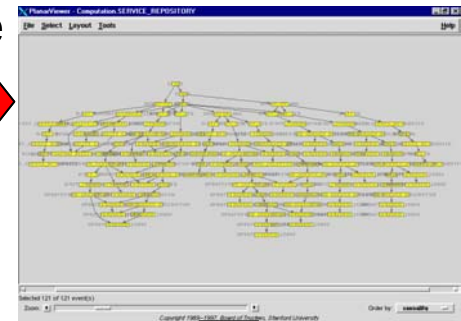
Topology

Scenario

Time	Command	Parameters
5	NodeFail	SM4
5	LinkFail	SCM1 SM4
10	GroupJoin	SM4 GROUP1
10	FindService	SU8 5 1 2 S XYZ ALL
50	AddService	SM4 SCM3 T ATT API GUI 20 30



Execute with Rapide



Analyze POSETs

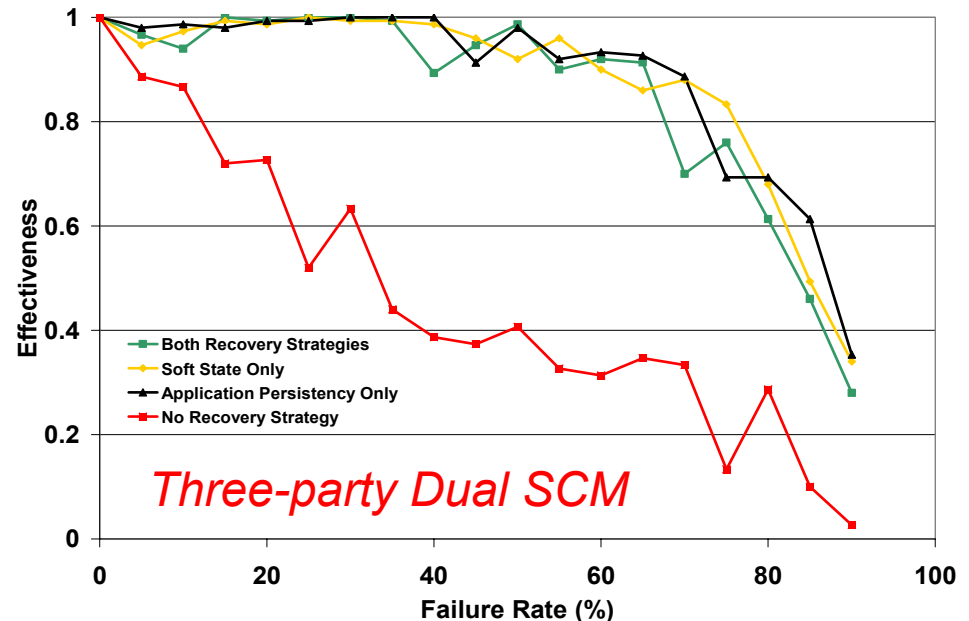
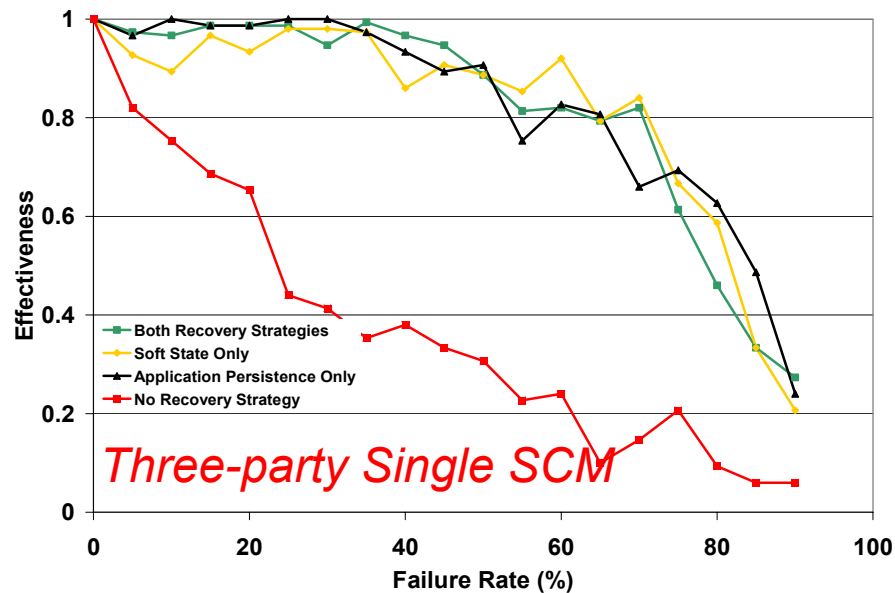
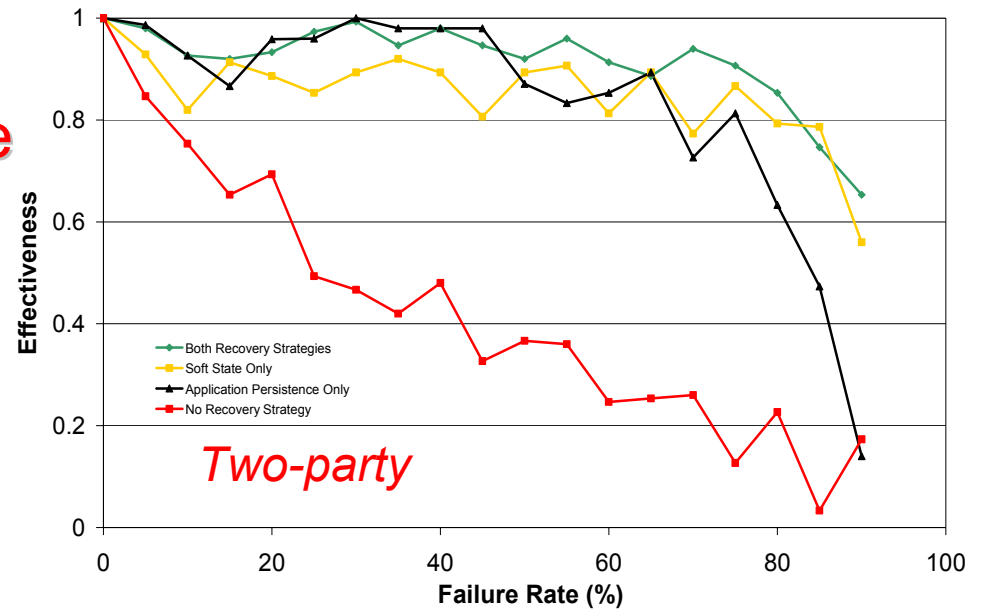
Consistency Conditions

- For All (SM, SD, SCM):  
(SM, SD) IsElementOf SCM registered -services  
Implies SCM IsElementOf SM discovered -SCMs (CC1)
- For All (SM, SD, SCM):  
SCM IsElementOf SM discovered -SCMs &  
(SD) IsElementOf SM managed -services (CC2)  
Implies (SM, SD) IsElementOf f SCM registered -services
- For All (SM, SD, SCM):  
SCM IsElementOf SM discovered -SCMs &  
(SM, SD) IsElementOf SCM registered -services &  
NOT (SCM IsElementOf SM persistent -list) (CC3)  
Implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf)
- For All (SM, SD, SCM, SU, NR):  
(SU, NR) IsElementOf SCM requested -notifications &  
(SM, SD) IsElementOf SCM registered -services &  
Matches((SM, SD), (SU, NR)) (CC4)  
Implies (SM, SD) IsElementOf SU matched -services

Use metrics to Assess Correctness & Performance

# Update Effectiveness in Response to Interface Failure

	Both Recovery Strategies	Soft State Only	Application Persistence Only	No Recovery Strategy
Two-Party Notification	0.915	0.853	0.836	0.431
Three-Party Notification Single SCM	0.819	0.816	0.828	0.383
Three-Party Notification Dual SCM	0.856	0.879	0.887	0.465

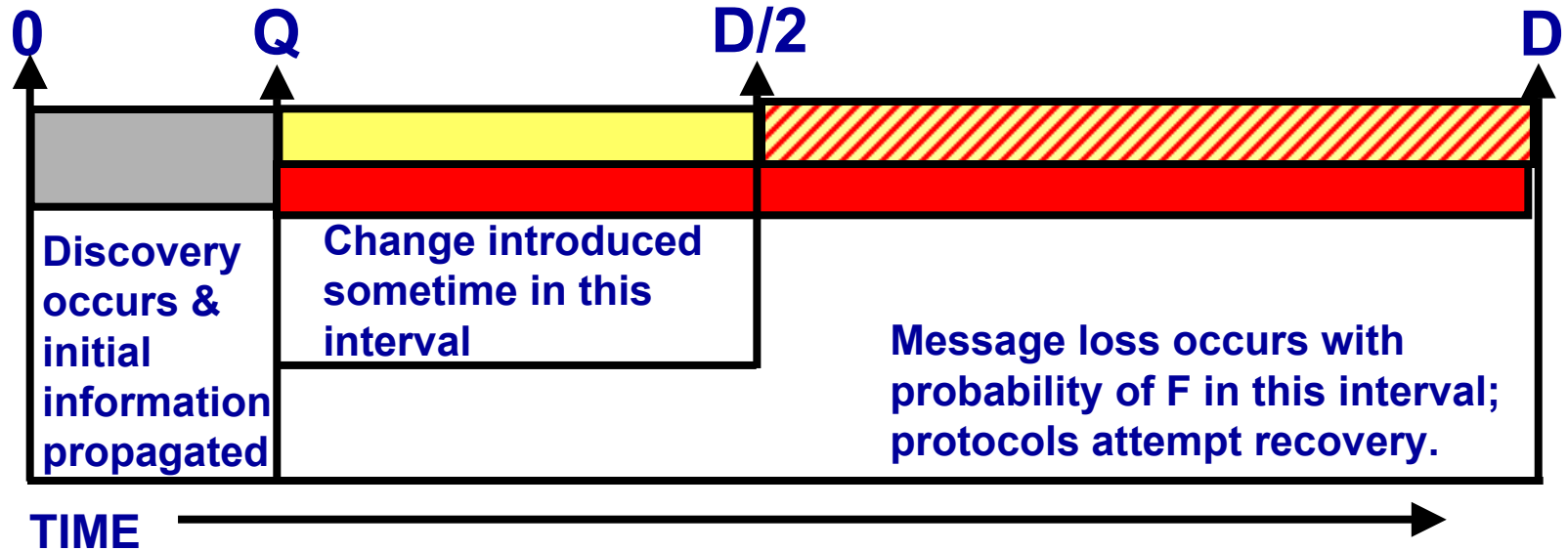


# Results

- **Under Conditions of Interface Failure**

- Performance decreases linearly in absence of recovery strategies
- Soft State alone :
  - In both architectures, discovery discard decreases time available to recover.
  - In two-party, Soft State recovery alone is insufficient because recovery is not stimulated when failures block Get Description Requests or Notifications, but not announcements.
  - In three-party, Soft State alone approaches performance of both strategies together, because discovery discarded after same period as when both strategies used together.
- Application Persistence alone:
  - In two-party, Application persistence may be sufficient, but in our experiments it's limited by lease renewal algorithm (residual 2.5% not renewed).
  - In three-party, Application Persistence performs as well as both strategies together because retries continue every 120s.
  - If additional SCMs provided, more paths for recovery and propagation allow Application Persistence to exceed both strategies together.

## Message Loss Model for Experiment



- Random Processes**
1. Choose a time to introduce the change [uniform( $Q$ ,  $D/2$ )]
  2. For each message transmission, determine if message is lost using  $F$

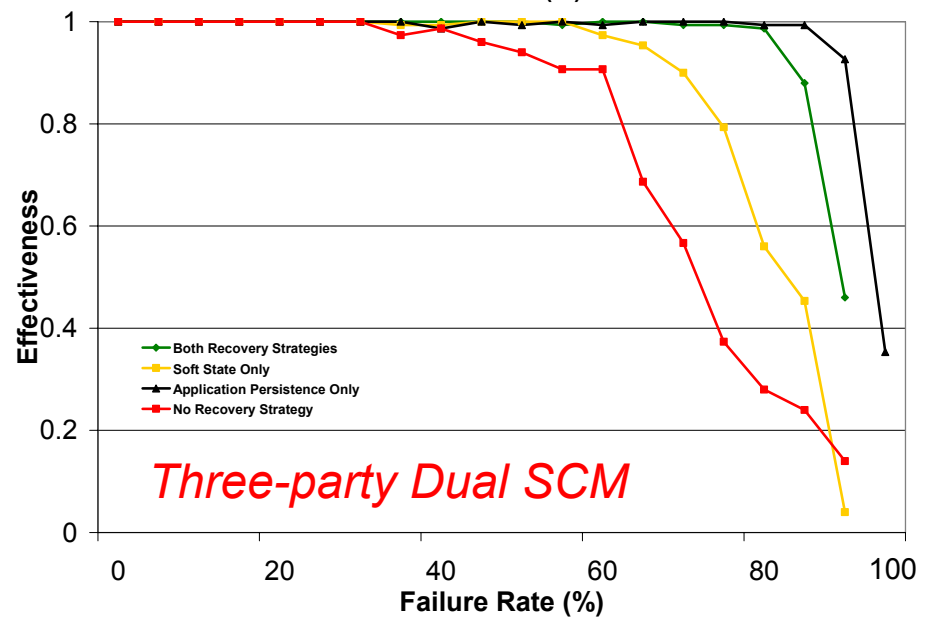
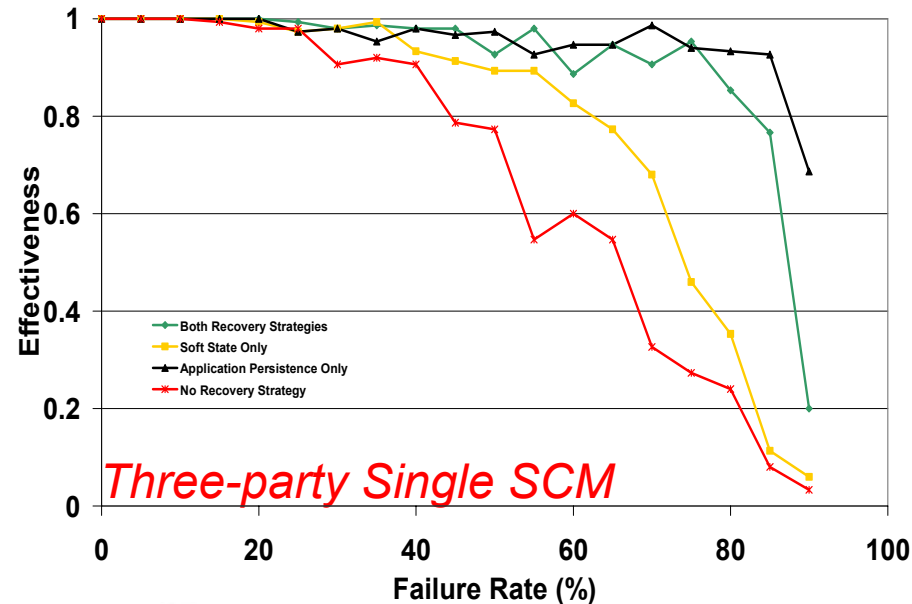
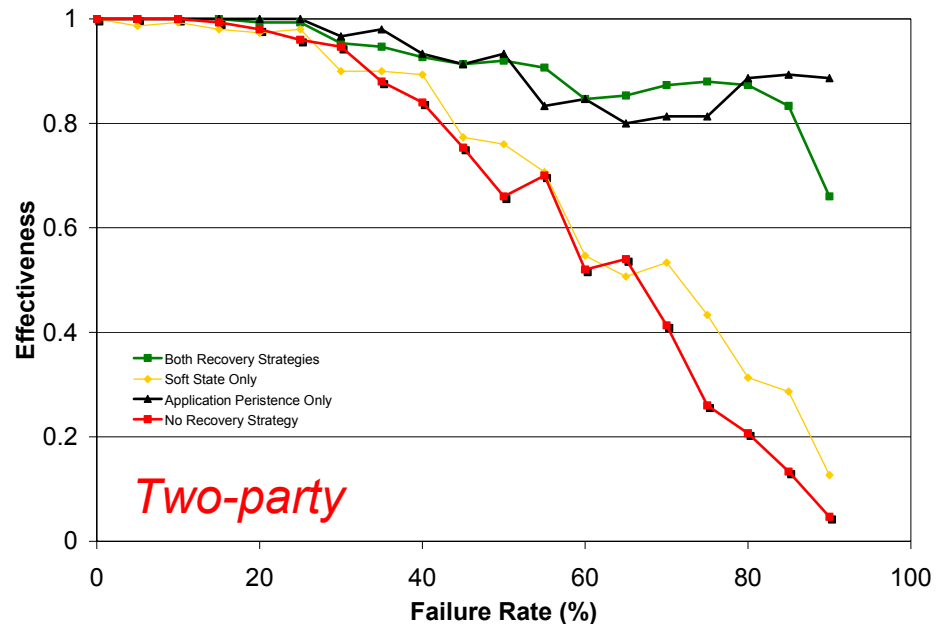
$Q$  = end of quiescent period (100 s in our experiment)

$D$  = propagation deadline (5400 s in our experiment)

$F$  = message loss rate (variable from 0% - 95% in 5% increments in our experiment)

# Update Effectiveness in Response to Message Loss

	Both Recovery Strategies	Soft State Only	Application Persistence Only	No Recovery Strategy
Two-Party Notification	0.914	0.715	0.921	0.675
Three-Party Notification Single SCM	0.913	0.781	0.954	0.679
Three-Party Notification Dual SCM	0.964	0.877	0.994	0.787



# Results

- **Under Conditions of Message Loss**
  - Again, performance decreases linearly without recovery strategy
  - In three-party architecture, additional SCMs provide more paths for propagation and recovery.
  - Soft State alone:
    - Performance under Soft State alone insufficient because after discovery discard, rediscovery messages continue to be subject to message loss (making it harder to rediscover at high failure rates).
  - In Application Persistence alone
    - Application Persistence better than both strategies together because retries continue every 120s AND additional messages for rediscovery are not used.
- However, if nodes fail and are replaced by new nodes (different experiment), Soft State becomes more important than Application Persistence.



# Observations on Self-healing in Distributed Systems

- Recovery strategies are critical for self-healing as failure rate increases.
  - More so than other factors (e.g., architecture, topology, consistency-maintenance mechanisms)
- Recovery strategies can interact in complex and unexpected ways
  - Redundancy (only one is necessary)
  - Complimentaryness (both are necessary)
  - Interference (one strategy prevents another from succeeding)
- When designing self-healing distributed systems based on service discovery protocols, need to consider:
  - The types of failure expected and their likelihood
  - Detailed protocol behaviors (e.g., discovery, update propagation, recovery) and not simply the application-programming interface.